(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
28 June 2001 (28.06.2001)

PCT

(10) International Publication Number
WO 01/46856 A1

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/AU00/01554

(22) International Filing Date:
18 December 2000 (18.12.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
PQ 4757 20 December 1999 (20.12.1999) AU

(71) Applicant *(for all designated States except US)*: YOURAMIGO PTY LTD [AU/AU]; 1st Floor, 80 Gilbert Street, Adelaide, South Australia 5000 (AU).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: STEELE, Robert, James [AU/AU]; 6 Canterbury Drive, Morpeth, New South Wales 2321 (AU). POWERS, David, Martin, Ward [AU/AU]; 72 Highland Drive, Bellevue Heights, South Australia 5050 (AU).

(74) Agents: WEBBER, David, Brian et al.; Davies Collison Cave, 1 Little Collins Street, Melbourne, Victoria 3000 (AU).

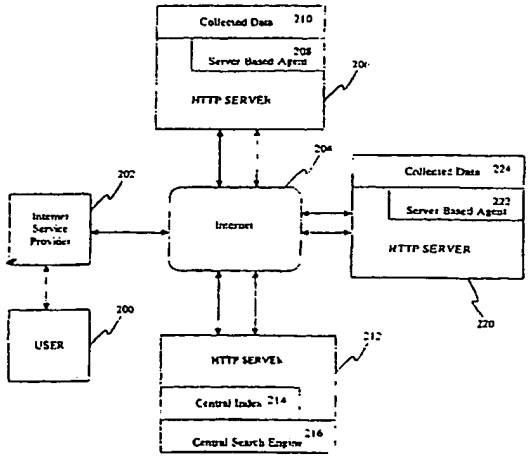
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: AN INDEXING SYSTEM AND METHOD



(57) Abstract: A method for generating an index of data available from a server, including processing data on the server to access data items for a central index, the data items including network addresses and terms, compiling an index file including the data items, and transmitting the index file to the central index. The processing may include locating database query statements in the data, and the data items then include input tuples for the statements. The index is accessible fro servers, and includes page entries including a program address for a program for generating a dynamic page and input tuples for submission to the program to generate the page, and search entries identifying the dynamic pages and identifying the tuples corresponding to search terms. A search engine operable on the index, is able to access the search entries to identify dynamic pages corresponding to search terms of a search query, and access the page entries to generate addresses for the dynamic pages identified, the addresses being generated on the basis of the program address and the tuples.

BNSDOCID: <WO___0146856A1_I_>

processing, and due to the volume of data to be processed it is common that a new or modified page will wait for several months before being processed.

Distributed indexers are available, such as Aliweb. In this system, the indexing information is manually entered into templates by the system administrator or the author of the page. The pages are then available to a spidering program for retrieval. Since the information about a page is generated by a human, the information about page content is usually very accurate. However, many administrators and authors are not prepared to provide such information, and those that are often do not spend sufficient time to complete the template, and so the index is frequently incomplete, and out of date.

In another type of search engine, such as that originally provided by Yahoo, the index is constructed by a manual inspection of pages by humans. Since the inspection is manual, the categorization of web pages under particular topics is generally fairly accurate, as are the ratings of the quality of the pages. However, the limited number of people available limits the extent to which the web is covered, and the rate at which new and modified web pages are reviewed.

Client based search engines, such as Fish, are based at individual searchers or web users. They offer greater scope for an agreeable user interface, and for personalized searching. However, they have the potential for wasting large amounts of bandwidth if independently searching a substantial portion of the web.

Some search engines, for example MetaCrawler and Dogpile, upon receiving a search request, search the search sites of other search engines, receive the results from these and consolidate the results for display to the user (this is known as a metasearch). This leads to better coverage of the web, since some search engines include data from sites not visited by other search engines. However, this is an inefficient approach, since there is considerable overlap between different search indices, there is also an additional delay in returning the results to the user, and methods available for ranking the results in a relevant order are limited.

The present invention also provides an agent having components for executing the steps of the method.

Preferably the method further includes:

receiving said index file at said central index, which has an index database; and

maintaining said index database on the basis of entries in said index file, said index database being adapted for use by a search engine

The present invention also provides an index of data accessible from servers, including:

page entries including a program address for a program for generating a dynamic page and input tuples for submission to the program to generate the page; and

search entries identifying the dynamic pages and identifying the tuples corresponding to search terms..

The present invention also provides a search engine operable on the index, including:

means for accessing the search entries to identify dynamic pages corresponding to search terms of a search query; and

means for accessing the page entries to generate addresses for the dynamic pages identified, said addresses being generated on the basis of said program address and said tuples.

The present invention also provides an indexing system including the agent, the index and the search engine.

The present invention also provides an indexing system, including:

a server for providing access to at least one site;

a server agent for creating an index file of data relating to the site; and

a central index for storing index information from the index file, wherein the server agent initiates communication with the central index for transfer of the index file.

FIG. 5 is a schematic illustrating an example of three web servers with server-based agents and a central index of the indexing system;

FIG. 6 is a flow diagram of the response of a server-based agent to receiving information that a document linked to one on another web sever has changed;

FIG. 7 is a flow diagram of a server-based agent notifying a web author that a document linked to on another web server has changed or moved;

FIG. 8 is a flow diagram of how a web browser may check with the central index to update changes to it's bookmarks;

FIG. 9A is an example of a web page with a form input for entry of a stock code and stockbroker's name;

FIG. 9B is the same example as FIG. 9A with entries made in the form input fields;

FIG. 9C shows the web page that is returned when the submit button in FIG. 9B is pressed;

FIG. 10 is a schematic illustrating how a dynamic page is created by entering data into a form page;

FIG. 11 is a schematic illustrating a web server with a server-based agent, static pages, and a database and from handling programs which create dynamic pages on request;

FIG. 12 is a flow diagram of a server-based agent process for indexing dynamic pages;

FIG. 13 is a block diagram of the central index;

FIG. 14 is a flow diagram of a process executed by the central index;

FIG. 15 is a flow diagram of a process executed by a central search engine of the indexing system for dynamic page indexing;

FIG. 16 is a flow diagram of a process executed by server-based agents for dynamic page indexing;

FIG. 17 is a diagram showing an example structure for static page index entries in the forward and inverted indexes of the central index;

FIG. 18 is a diagram showing an example structure for dynamic page index entries in the forward and inverted indexes of the central index;

114, and a display adapter 112. All of the computer components are connected by a system bus 115. The display adapter 112 may be connected to a display 116 for displaying a recommendation to a user. The user interface adapter 114 may be connected to a user input device 118. The user may be connected to the Internet through the network interface, 110. The user may also be connected to the Internet through an Internet interface 120, for example a connection through a modem to a service provider such as America Online ®, or through a cable connection to an Internet Service Provider.

In Figure 2, a user 200 is connected to an Internet service provider (ISP) 202 , who is coupled to the Internet 204. Also coupled to the Internet 204 is a first server, typically an HTTP Server (HS) 206 that implements and provides access to one or more sites which may contain, for example, html pages and other documents and scripts. The HS server 206 includes a server-based agent (SBA) 208 that carries out data collection activities from documents on the first HS server 206. The SBA 208 may operate at all times, or may run for intervals interspersed with periods of inactivity. The SBA 208 examines all documents located on the HS server 206 to create a listing of all document updates, stored in the collected data block 210. The SBA 208 transmits the listing of document updates in the form of an index file or index delta file to the central index (CI) 214, that can be provided by software executed on an index server 212. An index file is a file of data that provides information on what is contained on various locations on the network, such as what is contained on various web pages, together with addresses for locations of the data. The file may have a complex structure and be distributed over a number of files and/or servers. The reference to an index file may also include a reference to an index delta file, which is a file that simply contains changes that need to be made to an existing index file in view of changes to the data on the network. It will be apparent to those skilled in the art that an efficient implementation of the indexing system is to provide a software implementation of the SBAs and the CI and CSE, as described below, with the CI having the architecture described below with reference to Figure 13. It will also be apparent to those skilled in the art that a number of the components can be distributed in a communications network, such as the Internet, and also that a number of the software components can be substituted by

(b) **report (CI_address, index_delta)** -- makes a report to the CI, at address CI_address, transmitting the index_delta file that describes the local changes detected in the update operation.

The SBA 208 performs two basic indexing functions, namely updating its own index and the index_delta file, and reporting the index_delta file to the CI 214. These two basic indexing functions may operate with the same frequency. For example, the SBA 208 may have a timer that prompts the agent 208 into operation at regular intervals, e.g. every 24 hours. While not active, the SBA 208 maintains its index entries for all documents on the server from the previous active period. When the SBA 208 is awoken by its timer, it may check for changes in the directory tree of the server site, or if any individual files have been changed. Any changes that are detected are stored and then reported to the CI 214. Where there are no changes, the SBA 208 may be programmed not to send an index_delta file to the CI 214, so as to preserve bandwidth. The SBA 208 also updates its local index files for the documents at that server site.

Instead of updating at regular scheduled intervals, the SBA 208 may update whenever a page on the website is changed. In this case, the SBA 208 may offer to the author of the change (or to the website administrator) to submit the change to the CI 214 immediately. The SBA 208 may also prompt the author, in the same dialogue, to enter, or update, any subject keywords that the author feels reflects the page. These keywords may then be submitted to the CI 214 in the index_delta file. Of course, the author may simply ignore the option to enter keywords.

The time between report operations may be set at a fixed interval, or may be dynamically altered to fit the nature or needs of the local server 206 or the index server 212.

As part of the update operation, the SBA 208 may review hypertext links in the pages on the local server 206. The reviewed links are then compared with a link list formed during the previous update operation to determine whether links have been added or subtracted. The SBA 208 includes the list of changed links in the index_delta file transferred to the CI

(ii) an SBA sending an index_delta file to the CI based on its own parameter settings

(iii) an SBA sending a short message to the CI indicating that a change has occurred, so that the CI may retrieve the changes later in any manner which it chooses, including but not limited to sending a spider for indexing by existing methods, or for retrieving the index or index_delta file

(iv) an SBA sending a short message or an index or index_delta file to a regional index rather than to the CI

The CI 214 may also accumulate statistics relating to the load at local servers, in addition to accumulating indexing data.

The CI 214 may also have a list of construction and modification dates for all Internet documents. This date list may be helpful in searching, and allow a useful facility for obtaining the most up-to-date information on a topic. The CI 214 may also evaluate whether a site is one which is no longer maintained, based on information in the date list.

2.1 Stale Links

The CI maintains in the index database an index for each URL that lists the URLs of pages that include a link to it or reference it. This is a library of URLs that relates each subject URL to other URLs that have a page linking to the subject URL. When index information is reported to the CI indicating that a particular URL has been deleted or moved, the CI may search the URL index to determine which URLs contain links to the deleted URL, and then send notification to the SBA at each of the referring servers. The local SBA may then take some action in response to such notification. For example, the SBA may notify the authors of the referring page, or the website administrator, that the link has been deleted or moved. The SBA may also be programmed to take automatic action. One example of automatic action that the SBA may take in view of a deleted or moved link is to add a warning to the html code of the referring page to indicate that the marked link is no longer valid. Another example is that the SBA may replace the link with a link to the root directory of the site to which the URL had hitherto been referring, if possible. Where the

The SBA then identifies changes in links that have occurred since the previous link scan, at step 306. Those link tuples that relate to changed links are then stored in the **linkchange_list** file, at step 308. The SBA then proceeds to scan the remaining html documents on the server, at step 310, so that **linkchange_list** contains all changes in links contained in all the documents on the server.

The SBA may then proceed along different paths or may proceed along parallel paths. One option for proceeding, at step 312, is for the SBA to transmit to other servers the relevant changes contained in **linkchange_list**. This approach is illustrated in Figure 4, which shows three servers 410, 420 and 430 in communication through the Internet 440. The first server 410, having the server name **http: //www.server1.com**, has an SBA 412, entitled **SBA_server1**. An html document 414, entitled **doc1.html** is present on the first server 410. The second server 420 has the address **http: //www.server2.com** and also has an SBA 422 called **SBA_server2**. The second server 420 has two html documents, **doc2.html** 424 and **ref1.html** 425. The third server 430 has the address **http: //www.server3.com**. The third server 430 has an SBA 432 called **SBA_server3**. The third server 430 has an html document 435 entitled **ref2.html**.

The html document **doc1.html** 414 on the first server 410 includes a number of links, including a link to **ref1.html** 425 on the second server 420 and **ref2.html** 435 on the third server 430. The SBAs on each server, as described above at step 304, create a list of tuples of source and target documents. Therefore, the **linkto_list** file 416 created by the first SBA 412 includes at least the two tuples illustrated, i.e.,
(www.server1.com/doc1.html, www.server2.com/ref1.html) and
(www. server1.com/doc1.html, www.server3.com/ref2.html).

Likewise, the second server 420 also contains a **linkto_list** file 426 that lists tuples of source and target documents, where the source documents are html documents on the second server 420 and the target documents are documents referred to by links in the source documents and that are stored on other servers. In the illustrated example, **doc2.html** 424 includes a link to **ref2.html** 435 on the third server. Accordingly, the

The SBA then, at step 316, may transmit changes that have been detected in target html documents on that server, to those servers that contain the source html documents, i.e., those documents that include links to the current server. This permits servers of source, or **link_to**, documents to adapt the links in the source documents according to the changes in the target document. This is explained further below.

Another protocol for maintaining updated links, and avoiding stale links, includes the step 318 of transmitting the changes in the links to the CI. The CI maintains a list of all links between documents and informs source documents of changes in a target document when changes in the target document are received.

This is explained further with reference to FIG. 5, which illustrates first, second and third HTTP servers 510, 520 and 530 that contain html documents, and are connected via the Internet 540. The first server 510 includes an SBA 512 called **SBA_server1**, and the second and third servers, 520 and 530 also include respective SBAs 522 and 532. Each SBA assembles a **linkto_list** file 516, 526 and 536 in a manner according to step 304. The first, second and third servers 510, 520 and 530 are connected to a central index server 550, whose address is, for example, **http://www.ci.com**. The CI server 550 includes the CI 552. The CI contains indexing information received from the attached servers in a manner as described above.

The first SBA 512 transmits changes in links to the CI, at step 318. The CI 552 maintains a **link_list** 554 that is a listing of all links between documents. Therefore, since **doc1.html** 514 on the first server 510 includes links to **ref1.html** and **ref2.html**, **link_list** includes the tuple (**www.server1.com/doc1.html**, **www.server2.com/ref1.html**) and the tuple (**www.server1.com/doc1.html**, **www.server3.com/ref2.html**). Additionally, **doc2.html** 524 on the second server 520 includes a link to **ref2.html** 535 on the third server 530. Therefore, the file **link_list** 554 also includes the tuple (**server2.com/doc2.html**, **server3.com/ref2.html**). Since the CI 552 includes all link information, there is no requirement in this protocol to maintain a **linkfrom_list** file at each server.

If the SBA determines that the target document has not been deleted, nor been moved, the SBA then determines whether the contents of the target document have changed, at step 620.

One way to determine if the contents of the page have changed significantly is to check whether the title of the page has changed significantly. This involves comparing two small pieces of text. If substrings of the previous title remain, then the page has probably not significantly changed. If all the words are different, and there are no semantic links between the old and new words (a semantic dictionary such as wordNet can be used) then it can be concluded that the content has significantly changed.

Once the SBA has determined that the change in the contents of the target document is sufficiently important, at step 622, the SBA may then proceed in one or more different ways. For example, the SBA may simply inform the server administrator, at step 623, of the change in the contents of the target page. The administrator may then inform the author of the source page so that the author may make a manual determination as to whether the target page is still worthy of maintaining the link from the source page. The SBA may also be authorized or configured to remove the link in the source document, at step 624 or may be authorized to replace the link to the target document with a new link to an alternative target document, at step 626.

The method of choosing an alternative document is the same as for above case where a document has been deleted.

The SBA may also determine that the server on which the target document is located is no longer available, at step 628. If so, then the SBA may take one or more of the following steps. The SBA may inform the server administrator that the server containing the target document is no longer available, at step 630. The SBA may, if authorized or configured to do so, amend the source document, at step 632 with a mark indicating that the target document is no longer available. The SBA may also, if authorized or configured to do so,

maintained by the central index. The browser can be provided with an agent that periodically checks with the central index to ensure that the bookmarked pages are still available, at step 802 in FIG. 8. If the browser agent determines, at step 804, that the bookmark has disappeared, the browser agent may mark the bookmark as being invalid, step 806. If the browser agent determines from the central index that a bookmark has changed its URL, at step 808, then the browser agent may be configured to change the bookmark to reflect a new URL, at step 810. If the browser agent determines, at step 812, that the contents of the page located at the bookmark have been changed, the browser agent may recommend to the user an alternative bookmark, at step 814.

2.2 Dynamic Pages and Generated Static Pages

Dynamic pages are web pages that are not written by hand in html, typically the html that constitutes them is made by a program "dynamically". These pages are constructed by a program at the time at which, for example, an html form query is submitted by a user. Most dynamic pages use data from a database in their construction. Other kinds of dynamic pages (eg charts) are constructed by Java or Javascript programs or scripts. A dynamic page can be considered to be a web page that is generated by a program or script each time its URL is requested by a browser, and it is not stored on the web server. Most dynamic pages are generated from a backend database. Some pages may be generated in advance from a database and stored as static pages, and references to dynamic pages should usually be taken to include generated or pre-generated static pages. An input tuple is used to send inputs that results in the return of a valid dynamic page.

Some sites with information in databases do not create dynamic pages on request, but use a script to generate a large number of static pages which are updated periodically. These pages have similar characteristics to dynamic pages, in that in general there are unlikely to be links to them from anywhere else. Hence, conventional indexing methods such as spidering may not find these pages, because spiders find web pages via links. These generated static pages may be caches of frequently accessed dynamic pages rather than comprehensive sets of all possible pages. A generated static page or pre-generated static

SQL query, to the database. The cgi script then retrieves the requested data from the database, builds a page using this data and returns the dynamic page to the user. There are programs other than cgi scripts used to create dynamic pages, and any reference to a form handling program herein should be interpreted to mean any script or executable file of any kind or in any language, whether interpreted or compiled to machine code or intermediate or virtual machine code (including Java servlets and Remote Methods) that can be used to generate a dynamic page. A form page can be considered to be a web page that contains fields, menus and/or other means for a user to specify inputs, and is most often a static page, although the term also includes dynamic pages that contain fields to allow the user to initiate a new search, and other means of collecting inputs such as Java windows. An example illustrating how form pages are used to generate dynamic pages is described below with reference to Figures 9 to 10, when illustrating how the SBA operates on a server 206 as shown in Figure 11.

2.2.1 Overview of Indexing of Dynamic and Generated Static Pages

2.2.1.1 Operation of the SBA

The general steps executed by an SBA to index dynamic pages is shown in Figure 12. During installation, the SBA may be configured to suit the needs of the web site administrator, at step 1202. For example, the SBA may be configured with a number of parameters, including:

(i) how often the SBA checks the website for changes

(ii) automatic updating of stale links, or alerts only to the website administrator with suggested updates

(iii) restrictions on pages which are not to be indexed

(iv) restrictions on data (eg a column) in any database(s) which is not to be indexed (eg confidential information, information that is not important or not useful, or information that is only to be displayed if other information is provided by a user - an example of this last point is where a user may enter a person's name, and the phone number will be returned, and this is not to occur the other way around.

maintained that can comprise a dictionary and/or thesaurus of indexed terms of the inverted index 1306. It will be apparent to those skilled in the art, that the functions of the lexicon 1308 could be integrated into the inverted index 1306.

The lexicon 1308 provides word, or term, to wordID mapping, the inverted index 1306 provides wordID to docID mapping, and the forward index 1302, 1304 provides docID to URL mapping. When a searcher provides one or more query terms for which to search, the lexicon 1308 is consulted to determine whether or not the terms exist in the index, and if they do, their corresponding wordID. The inverted index is then searched to find all docIDs for the wordIDs, representing the complete set of documents that contain the searcher's query terms.

Information stored in the inverted index 1306 may include:

(i) whether the docID corresponds to a static or dynamic page;

(ii) whether the query terms are found in an important field (such as a title) or the body of the text;

(iii) whether the query terms are closely located (and hence possibly related) within a given document, or apparently unrelated; and

(iv) whether the document contains all, or only some, of the query terms.

This information is used to order the list of docIDs such that those documents most likely to be relevant to the user's query are at the head of the list. Then for each docID, the forward index is consulted to determine the URL through which the original document can be retrieved. In the case of a dynamic page, this involves reconstructing the URL from the form handling program URL, method, input fields and input tuple. The forward index content may also be used to confirm the presence of exact phrases that the searcher may have specified, and to provide short extracts of the relevant content that the searcher may review when considering which results to pursue.

With reference to Figure 14, the CI waits to receive a message from an SBA at step 1402. When the CI receives a message, it then adds new indexing information and modifies existing indexing information in its forward index 1302, 1304, at step 1404. The CI then

2.2.2 Indexing of Dynamic Pages

2.2.2.1 Stockadvice Example

The indexing of dynamic pages is described below with reference to a stockadvice example whereby dynamic pages can be generated, as shown in Figures 9A to 9C, in order to provide recommendations from stockbrokers on companies listed on various stock exchanges. The form page is shown in Figure 9A, and the form page with a query entered by the inclusion of specific inputs is shown in Figure 9B. The resulting dynamic page is then generated and sent to a user's browser as shown in Figure 9C with the highlighted entries 900 representing the data extracted from the database 1002 of a web server 206, as shown in Figure 10. Figure 10 illustrates the message flow whereby the form page 1004 sent to a user's browser 1006 can be used to return a http request to a form handling program 1008 that receives, with the request, data entered in input fields 1010 of the page 1004. The program 1008 generates an SQL query to the relational database 1002 in order to return a result to the program 1008 that can then be used to generate a http response back to the user's browser. The response includes the code defining the dynamic page generated by the program 1008, and the response causes display of the dynamic page, as shown, in Figure 9C on the user's browser.

More specifically, the URL for the form page of Figure 9A is:

http://www.stockadvice.com/broker.html

One input field is for a stock code and the other input field is for the name of a broker. The name of the form input for stock code is 'scode', and the name of the form input for broker name is 'bname'.

An example of a HTML form tag on broker.html could be:

<FORM ACTION = "http://www.stockadvice.com/cgi-bin/brokerdata"

stock_info table

STOCK_CODE	EXCHANGE	STOCK_NAME
AOL	NY	America Online
MSFT	NASDAQ	Microsoft Corp
YHOO	NASDAQ	Yahoo
CNN	NY	CNN

A third table, **stock_brokers**, has the columns **name** (the primary key), **phone_no** and **email**.

stock_brokers table

NAME	PHONE_NO	EMAIL
smith	712349876	smith@aol.com
jackson	598765432	jackson@aol.com
andrews	124683579	andrews@aol.com

The cgi-script of the program 1008 generates the following SQL query, when a user enters 'YHOO' and 'jackson' into the respective from fields, and then hits the submit button:

```
select rating, stock_name, email
from stock_info, stock_rating, stock_brokers
where
        stock_info.stock_code='AOL' and
        stock_brokers.name='jackson' and
        stock_info.stock_code= stock_rating.stock_code and
 stock_brokers.name = stock_rating.name;
```

The SBA also needs to determine the location of the Web Site (WS) (ie the actual files that can be served to browsers). This may be provided manually by a computer administrator at step 1608, or the SBA can automatically deduce this at step 1610. One way of accomplishing this is for the SBA to look through the HSDT to find a sub-tree of this which contains html files.

The SBA then looks at each page in the WS to determine if it can possibly be used for making dynamic pages, at step 1612. The SBA makes a list, **dyn_list** of those pages that could make a dynamic page. One way of determining this is by identifying the presence of an html form in the page. In our Stockadvice example, the SBA would look at the file "broker.html" and would recognize that it has a form tag in it. It would then add this file to **dyn_list**.

The SBA then creates a new list called **dyndb_list**, at step 1614. The list **dyndb_list** generally will contain the same number or less items than **dyn_list**, because **dyndb_list** only contains a list of those pages that can create dynamic pages via accessing a database. For pages in **dyn_list**, the SBA looks at the file that is named as the form action file, **act_file**. If **act_file** is a binary (compiled to native machine code or some form of interpreted byte code, eg for the Java Virtual Machine), then the SBA extracts a list of strings, **str_list**, out of it. A string is a sequence of bytes found in a binary that correspond to values that represent characters such as letters, digits and punctuation. So **str_list** will typically be a list of readable words and names or terms. If **act_file** is not a binary, then the strings are readily visible. The SBA then looks through the members of **str_list** to find a string that is an SQL statement. In particular the SBA will be looking for an SQL select statement. An SQL select statement is used for querying a database. These pages whose **act_files** contain an SQL select statement will be assumed to be those that access a database to construct dynamic pages. These pages will be recorded in **dyndb_list**, and the names of their action files stored in **dbact_list**. In the Stockadvice example, **act_file** is cgi-bin/brokerdata. As this form handling program contains an SQL select statement, the file broker.html will also be in **dyndb_list**.

The SBA deduces which vendor's database is being accessed by each entry of **dyndb_list**, at step 1622. Usually only one database will be visible to cgi-scripts at the WS, but not necessarily. If there is only one database, then it is known that each page in **dyndb_list** accesses it. If there is more than one database, then the SBA determines which pages from **dyndb_list** access which database. To determine this the SBA uses heuristics for how certain cgi scripts connect to particular databases. For example, if a cgi script is compiled C, and it accesses an Oracle database, then one of the strings extracted from the cgi script will contain the library name, "sqlca". Once the match from cgi script to database is made, the SBA can proceed to identify the database name, the username and password for the database, and its network address if that is required.

The SBA now determines the name of the database and the username and password for the database, if these are required. At steps 1624 and 1628 this may be performed manually by the site administrator entering these details. Alternatively, the SBA can automatically determine these at steps 1626 and 1630. The database name is generally included in an environment variable for the database being accessed, or may be found in the cgi-script itself, or in some other system information source. For example, with Oracle, if the ORACLE_SID environment variable is set, it will contain the database name. If not, it is likely to be at the end of the username string. The SBA uses heuristics for each combination of database type and cgi-script type to look for the database name. For the Stockadvice example the database name will be found in ORACLE_SID, since cgi-bin/brokerdata accesses an Oracle database.

The SBA uses similar heuristics to look for the username and password for the database, at step 1630. These will also be in environment variables, in the script itself, or in some system information source. The SBA uses heuristics for each combination of database type and cgi-script type. For the Stockadvice example the username and password are contained in the text of the form handling program cgi-bin/brokerdata, and the SBA would identify which strings in the program are most likely to be a username and password, and it would then try them to confirm that they were correct.

Enter code: <INPUT TYPE = "text" NAME = "code" SIZE = 10>
<INPUT TYPE = "submit" VALUE="Get quote!">

The query is:

```
select stock_name, close
from stock_info, stock_prices
where
 stock_info.stock_code = :code_that_was_input and
 stock_info.stock_code = stock_prices.stock_code
```

This assumes the cgi-script is compiled C – :code_that_was_input appears to be a program variable and this would match to the input field called "code". This shows that this input field corresponds to the database column stock_info.stock_code. This database column contains the set of all possible inputs to the form.

If there is more than one input field, matching input fields to database columns is more difficult. This matching may be achieved by the SBA making a request to the cgi-script (in the process specifying values for each form input field) and observing the resultant input to the database. This observation may be carried out in a number of different ways. One way is for the SBA to install an Open Database Connectivity (ODBC) sniffer or some other method, as shown at step 1634. This will detect and log queries being sent to a particular database. There are other methods that will work for a broader range of operating systems or cgi-script types. Step 1636, one of these alternatives, involves the SBA altering the form handling script (if in uncompiled form) or providing a wrapper (if compiled) so that it informs the SBA of SQL queries that it carries out.

After the SBA has monitored SQL queries (step 1634 or 1636), it then determines the relationship between form inputs and database columns by one of two methods. At step 1638, the SBA makes queries to the form interface for each page, observes the resultant SQL query and notes which form inputs match which entries in which database column.

approach is a variant of this latter alternative, namely, to send tuples of information extracted from a database (Cases 1.1, 2.1, 3.1 and 4.1). This "tuple method" is very generally applicable, and is useable even when there is no direct access to the database, although in the rare case of joins of independent tables Cases 1.2 or 2.2 will be more efficient, as discussed previously, and may be automatically selected by the SBA.

The different methods are summarised as follows:

Case 1: Index the inputs of all possible dynamic pages

- Case 1.1: The SBA determines and sends all possible input tuples (step 1658)
- Case 1.2: The SBA sends all database columns used by the form handling program to extract data from the database used to create the dynamic page (eg SQL "where" clause (step 1660))

Case 2: Index the full-text of all possible dynamic pages

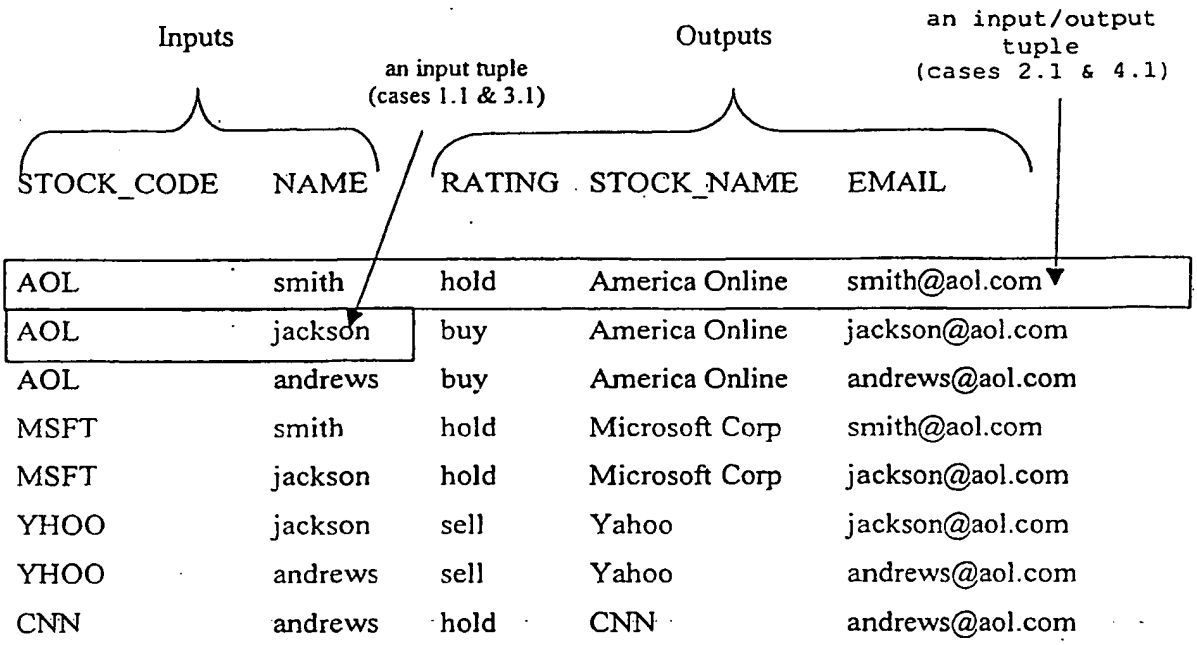
- Case 2.1: The SBA determines and sends the static text and all possible input/output tuples (step 1662)
- Case 2.2: The SBA sends the static text and all database columns that:
  - (i) are used by the form handling program to extract data from the database used to create the dynamic page, and
  - (ii) are output columns, (step 1664)
- Case 2.3: The SBA generates an index of all possible dynamic pages as if they were static pages

Case 3: Index the inputs of pages that have been retrieved by human searchers (step 1648)

- Case 3.1: The SBA observes user inputs entered by human searchers and sends input tuples

Case 4: Index the full-text of pages that have been retrieved by human searchers (step 1650)

giving the following input/output tuples:



STOCK_CODE	NAME	RATING	STOCK_NAME	EMAIL
AOL	smith	hold	America Online	smith@aol.com
AOL	jackson	buy	America Online	jackson@aol.com
AOL	andrews	buy	America Online	andrews@aol.com
MSFT	smith	hold	Microsoft Corp	smith@aol.com
MSFT	jackson	hold	Microsoft Corp	jackson@aol.com
YHOO	jackson	sell	Yahoo	jackson@aol.com
YHOO	andrews	sell	Yahoo	andrews@aol.com
CNN	andrews	hold	CNN	andrews@aol.com

Any static text that is put on the dynamic page by the form handling program is noted at step 1666. These will generally be the same and thus may be sent once for the whole set of dynamic pages from a dynamic page creation point. Searches may also be performed on these words. The static text, the above database content (ie input/output tuples) and information such as the URL of the form-handling program are sent to the CI at step 1668.

In some sites there will be static pages that have been pre-generated from a database to speed up access by avoiding database retrieval when the page is requested. These static pages should not be indexed separately, but rather the SBA indexes them using the tuple method in the same way as for dynamic pages.

The generation of these pages is similar to SBA indexing Case 2.3. The difference is that Case 2.3 generates and indexes dynamic pages, but does not store them, while the above sites generate the pages and store them as static pages for fast access. Therefore, these

queries that are made to the form interface at step 1648. For efficiency reasons the SBA only sends the observed input tuples after it has observed a number of them. This is done at step 1652.

The advantage of this approach is that the information being indexed at the CI is being filtered for its usefulness – it will only be indexed if it is being used.

One approach is where the SBA installs an ODBC sniffer (earlier steps have possibly already required its installation). This ODBC sniffer watches the inputs coming into a backend database from a form handling program. Earlier upon inspecting the form handling program, the SBA will have seen the words in the SQL query, so that it will be able to recognize it again. The SBA will have also worked out the mapping from form input fields to variable slots in the SQL query. This means when the SBA sees the SQL query at runtime it will be able to recognize what values were placed into form inputs.

In addition, the SBA extracts static text that appears on the dynamic pages. The SBA may also send the word positions of these words and also the word position at which words extracted from the database are inserted, to allow phrase searching.

The SBA then needs to check that the query returns a non-empty result set, by re-executing the query itself or some other method. If the result set was non-empty the SBA can report this input tuple to the CI.

The SBA may keep just a list of queries, and only send the top few, above a threshold, to the CI. Alternatively, the SBA may report a query to the CI after its incidence passes a certain preset threshold.

Case 4.1

This case is implemented in steps 1650 and 1654. This case is very similar to Case 3.1. The only difference is that the SBA, upon seeing a user query to the form interface, also

One method for achieving this is by using database triggers. The SBA may install database triggers that communicate with it when columns which affect the indexing information of dynamic pages change.

A typical database allows only one trigger of a particular type per database table. For this reason it is convenient for the SBA to construct a single trigger per table that notifies the SBA of changes relevant to any dynamic page creation points that make use of that table, at step 1670. At step 1672, the SBA installs triggers that have been constructed in all databases used by entries in **dyndb_list**.

The following is an example trigger that might be installed on the **stock_info** database table in the Stockadvice example. This would be the trigger used if full-text indexing was being used (ie Cases 2 and 4). If just inputs were being indexed then only the **stock_code** column would need to be checked. There is also a need for triggers on the other database tables in the Stockadvice example.

```
CREATE or REPLACE TRIGGER sender0
AFTER INSERT or UPDATE OF  stock_code, stock_name
ON  stock_info
FOR EACH ROW
DECLARE
 Msg        varchar(30);
 PipeName   varchar(30);
 LastStatus  integer;
 TraceMode   varchar(30);
BEGIN
 PipeName   := 'Slavko';
 LastStatus := 0;
 if (:new.stock_code is not null) then
    Msg := Msg ||':stock_info:'||'stock_code:'|| :new.stock_code;
 end if;
```

stock_brokers.name = stock_rating.name;

giving the result:

STOCK_CODE	RATING	STOCK_NAME	EMAIL
AOL	buy	America Online	jackson@aol.com
MSFT	hold	Microsoft Corp	jackson@aol.com
YHOO	sell	Yahoo	jackson@aol.com

The SBA calculates affected tuples for all form handling programs that access the particular database column that is changed. The CI locates all the old tuples and removes or updates them.

2.2.2.6 Structure of the CI 214 and behaviour of the CSE 216

The basic structure of the CI includes a forward index and an inverted index. For static pages, information is stored in these indexes using established techniques as used by existing search engines. Examples of forward index entries 1702 and inverted index entries 1704 for static pages are shown in Figure 17, together with example entries 1706 for the Lexicon (the notation "field:x" denotes x bits for the field). The format of the hits may comprise 16 bits, with 12 bits for the position of a word or term in the documents and 4 bits for other information, such as text capitalisation. The structure used by the CI for forward index entries 1802 for dynamic page creation points, involving the storage of tuples, as in Cases 2.1 and 4.1, is shown in Figure 18, with inverted index entries 1804 for Cases 2.1 and 4.1, and Lexicon entries 1806 used with the inverted index. In this case, the format for a hit is simply the tuple number, which may be stored using 16 bits.

Figure 18 is an example of a CI structure using forward and inverted index entries for indexing dynamic pages. The forward index contains blocks of information about whole dynamic page creation points. Each of these blocks is indexed with a number (dynID).

For Cases 1.2 and 2.2 a hit in the inverted index refers to the column and row in which the word occurs in the forward index.

A hit for a static page, or the static text on a dynamic page as in cases 2.3 or 4.2, is simply its word position on the page, plus other information like text attributes, as shown in Figure 17.

2.2.2.8 Interaction Between the CSE and CI for the Stockadvice Example

The data that is sent to the CI in the Stockadvice example for Case 2.1 or 4.1 has the following tuples:

STOCK_CODE	NAME	RATING	STOCK_NAME	EMAIL
AOL	smith	hold	America Online	smith@aol.com
AOL	jackson	buy	America Online	jackson@aol.com
AOL	andrews	buy	America Online	andrews@aol.com
MSFT	smith	hold	Microsoft Corp	smith@aol.com
MSFT	jackson	hold	Microsoft Corp	jackson@aol.com
YHOO	jackson	sell	Yahoo	jackson@aol.com
YHOO	andrews	sell	Yahoo	andrews@aol.com
CNN	andrews	hold	CNN	andrews@aol.com

All the tuples will be stored as they are in the forward index. The CSE notes which of the columns in the tuples correspond to form inputs – the first two in the case of the Stockadvice example.

A user is able to perform a search for 'jackson' AND 'Yahoo' (where AND is a boolean operator). The CSE recognizes from its inverted index entry, the tuples in which at least one of these words occur. A pre-results list contains all dynIDs that have any tuples

2.3 Indexing Applet Pages

Applets are another method commonly used to access databases. An applet is a small program written in Java. An applet can be named in the html code of a web page, and when a user views that web page, the applet, along with the web page are downloaded to the user's computer. The applet program is then automatically run. The applet can present boxes for entering text on the web page, and buttons to click to submit this text. Upon submission, the applet can generate a change in appearance on the page, to show the results of the query.

These pages are not dynamic pages in the sense discussed previously, but they do have the feature of being able to accept user input and then display results drawn from a backend database.

There are two main ways that a database may be accessed from an applet. First, the applet may contain the statements for connecting to a database itself. The other main option is that the applet does Remote Method Invocation on another Java class which in turn carries out the database access.

2.3.1 Server Based Agent for Applets

When the SBA is first installed it looks though the web site's directory tree, checking each html page. As described previously, an SBA is able to identify pages that contain a form tag. In this case, it also looks for pages that contain an applet tag.

To narrow these pages to applets that only access databases, the SBA looks at the ascii strings from the Java binary, and identifies an SQL query. If the applet itself accesses the database the SBA should find an SQL query in the strings from the applet. If an SQL statement is found, then this page is considered to be one whose output involves accessing a database.

The SBA then considers the SQL query used by the applet, as this was found earlier to verify that the applet accesses a backend database.

The process of matching text fields to database columns is the same as for form handling programs. The SBA installs an ODBC sniffer, which monitors queries sent to the backend database. The SBA then sends a query to the applet and monitors the SQL query that is produced.

The structure of the CI and the information sent to the CI from the SBA is generally the same as for the dynamic page case. This information includes the input tuples or database columns corresponding to text field inputs.

In order to create a clickable link on the search engine's results page, the SBA installs a new applet at its site that is a subclass of the original applet. The SBA also installs a new page that includes the new applet.

The link that appears on the search engine's results page will be to another script installed by the SBA. This script reads the query string sent by the search engine, and alters the applet tag on the page that includes the new applet by including the query string that has been sent by the search engine as a parameter. The script then redirects to the page that includes the new applet. The new applet is a subclass of the original. In its init() method it will read in the query string, set the text in the text fields accordingly and mimic the clicking of the submit button. An example of part of this code follows:

```
public class ResultApplet extends DBApplet{

 Button submit = new Button("Enter");

 public void init(){
  super.init();
  field1.setText( "sometext");
```

(iii) rank search engine results according to average time users spend on a page, demographic profile of the user, and/or keywords used in previous searches which found a particular page.

2.4 E-Commerce Applications

E-commerce is a specialist application of an SBA providing indexing information on dynamic pages. A CI may be used to provide an e-commerce portal for SBAs installed at shopping sites. Information that may be indexed includes:

(i) product name/model/manufacturer etc
(ii) price, including quantity pricing and discounts, taxes, etc
(iii) location
(iv) delivery time and freight cost options
(v) quality and reviews
(vi) picture of the product
(vii) warranty information
(viii) payment options, loyalty programs, etc

The E-commerce portal may also provide searching based on one or a combination of criteria, such as price including freight, delivery time, warranty period and location, quality and reviews and payment requirements.

Purchases may occur either through the CI's portal or a user may be directed to the relevant e-commerce site.

The CI's e-commerce portal may perform total cost calculations to identify an optimal cost based on the location of the buyer and the physical location of the product (freight costs), sales tax and duty issues, etc.

CLAIMS:

1. A method for generating an index of data available from a server, including:

processing data on said server to access data items for a central index, said data items including network addresses and terms;

compiling an index file including said data items; and

transmitting said index file to said central index.

2. A method as claimed in claim 1, wherein said processing includes determining changes in said data items, and said index file is an index delta file comprising said changes in said data items.

3. A method as claimed in claim 1, wherein said processing includes locating database query statements in said data and said data items include input tuples for said statements.

4. A method as claimed in claim 3, wherein said data items include additional data for accessing a database corresponding to said statements.

5. A method as claimed in claim 4, wherein said additional data includes the network address of a form handling program.

6. A method as claimed in claim 5, wherein the additional data includes the network address of a form page, and details on the input fields for the tuples and columns of the database.

7. A method as claimed in claim 6, wherein the additional data includes terms from the form page and terms from dynamic pages generated by the form handling program in response to tuples submitted on the form page.

17. A method as claimed in claim 2, wherein said determining includes installing database triggers to detect changes in columns of a database accessible by query statements in said data, said changes including said changes in said columns.

18. A method as claimed in claim 2, wherein said processing includes generating link pairs from said data, said link pairs including a source network address and target network address and said data items include said pairs.

19. A method as claimed in claim 18, including detecting a change in at least one of said pairs, and sending a change notification to a location corresponding to said source address of said pair.

20. A method as claimed in claim 19, including receiving said change notification and adjusting said data associated with said source address on the basis of said change.

21. A method as claimed in claim 20, wherein said adjusting includes replacing tags with said target address with tags to a new target address.

22. A method as claimed in claim 18, including detecting a change in at least one of said pairs, and sending a change notification to a location corresponding to said target address.

23. A method as claimed in claim 22, including receiving said change notification and establishing a link referral page on the basis of said change at a server associated with said target address.

24. A method as claimed in claim 1, wherein said processing includes accessing statistical data on said server, such as relating to data requests received at said server and data responses sent from said server, and said data items include statistical data.

33. A search engine operable on the index claimed in claim 32, including:

means for accessing the search entries to identify dynamic pages corresponding to search terms of a search query; and

means for accessing the page entries to generate addresses for the dynamic pages identified, said addresses being generated on the basis of said program address and said tuples.

34. An indexing system including:

an agent as claimed in claim 27;

an index as claimed in claim 32; and

a search engine as claimed in claim 33.

35. An indexing system, including:

a server for providing access to at least one site;

a server agent for creating an index file of data relating to the site; and

a central index for storing index information from the index file, wherein the server agent initiates communication with the central index for transfer of the index file.

36. An indexing system as claimed in claim 35, wherein the server agent is adapted to review the at least one site and compile an index delta file, representing changes to the at least one site, the delta file being transmitted to the central index for updating the index information.

37. An indexing system as claimed in claim 36, including a plurality of servers and associated server agents arranged to transmit a respective index file and/or index delta file to the central index.

38. An indexing system as claimed in claim 37, wherein each delta file includes information on any change affecting the validity of links in the sites of the respective servers.

46. An indexing system as claimed in claim 43, wherein the server agent associated with the dynamic page introduces a trigger in the database to identify any changes in the database, the changes being included in the index delta file associated with the dynamic page.

47. An indexing system as claimed in claim 43, wherein the index file associated with the dynamic page includes a record of inputs of pages previously retrieved therefrom by users.

48. An indexing system as claimed in claim 43, wherein the index file associated with the dynamic pages includes an index of the full text pages of the dynamic pages, previously retrieved by users.

49. An indexing system as claimed in claim 43, wherein the index file associated with each dynamic page for which input/output tuples or row/column indexing is stored further includes an identifier for identifying the form handling program and text associated with the page.

50. An indexing system as claimed in claim 43, including a server for accessing the central index in response to search queries.

51. A method of indexing, including:

providing a server agent for indexing sites provided by a server;

compiling an index file representing site data of the sites; and

transmitting the file to a central index, wherein the server agent initiates communication with the central index for transfer of the index file.

52. A method of indexing as claimed in claim 51, wherein the server agent processes the sites and compiles an index delta file, representing changes to the one or more sites, the delta file being transmitted to the central index for updating index information held by the central index.

60. A method of indexing as claimed in claim 59, wherein an index file includes possible outputs used to generate the dynamic pages.

61. A method of indexing as claimed in claim 58, wherein the server agent associated with a dynamic page identifies at least one database used to create the dynamic page, and extracts the text and input tuples for inclusion in the index file.

62. A method of indexing as claimed in claim 61, wherein the server agent extracts output tuples associated with said input tuples.

63. A method of indexing as claimed in 62, wherein said server agent extracts input and/or output columns of the database for inclusion in the index file.

64. A method of indexing as claimed in claim 61, wherein the server agent introduces a trigger in the database to report to the server agent when changes occur in the database, the changes being included in an index delta file.

65. A method of indexing as claimed in claim 58, wherein the index file includes a record of inputs and/or outputs used to generate previously retrieved dynamic pages.

66. A method of indexing as claimed in claim 58, wherein the index file includes an index of the text of dynamic pages previously retrieved by users.

67. A method for indexing dynamic pages including:
identifying at least one database accessed in producing a dynamic page;
determining the parameters and environment variables of the database;
determining a relationship between input fields of the page and the database;
identifying columns of the database that correspond to inputs; and
storing data of the columns in an index file.

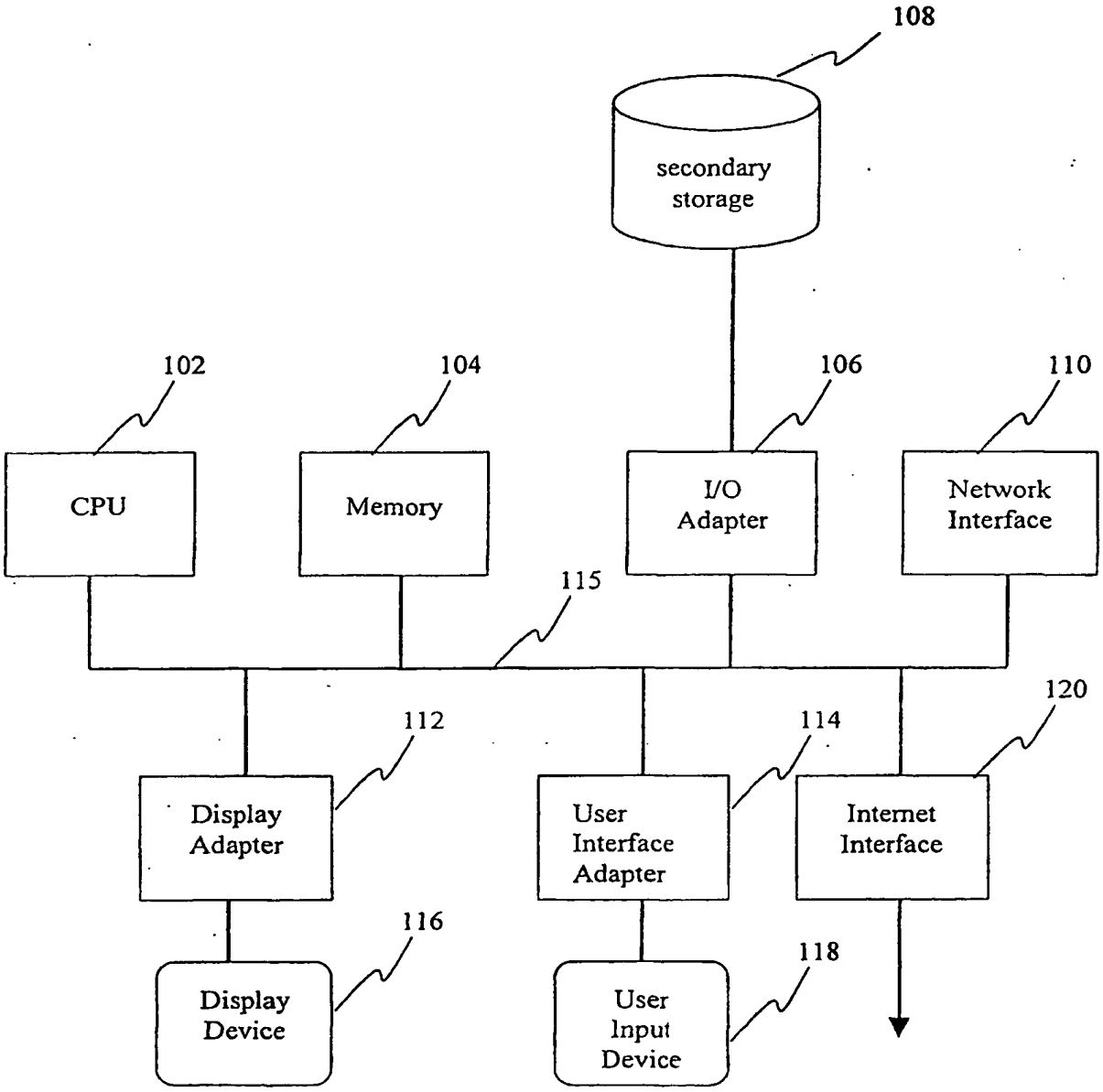
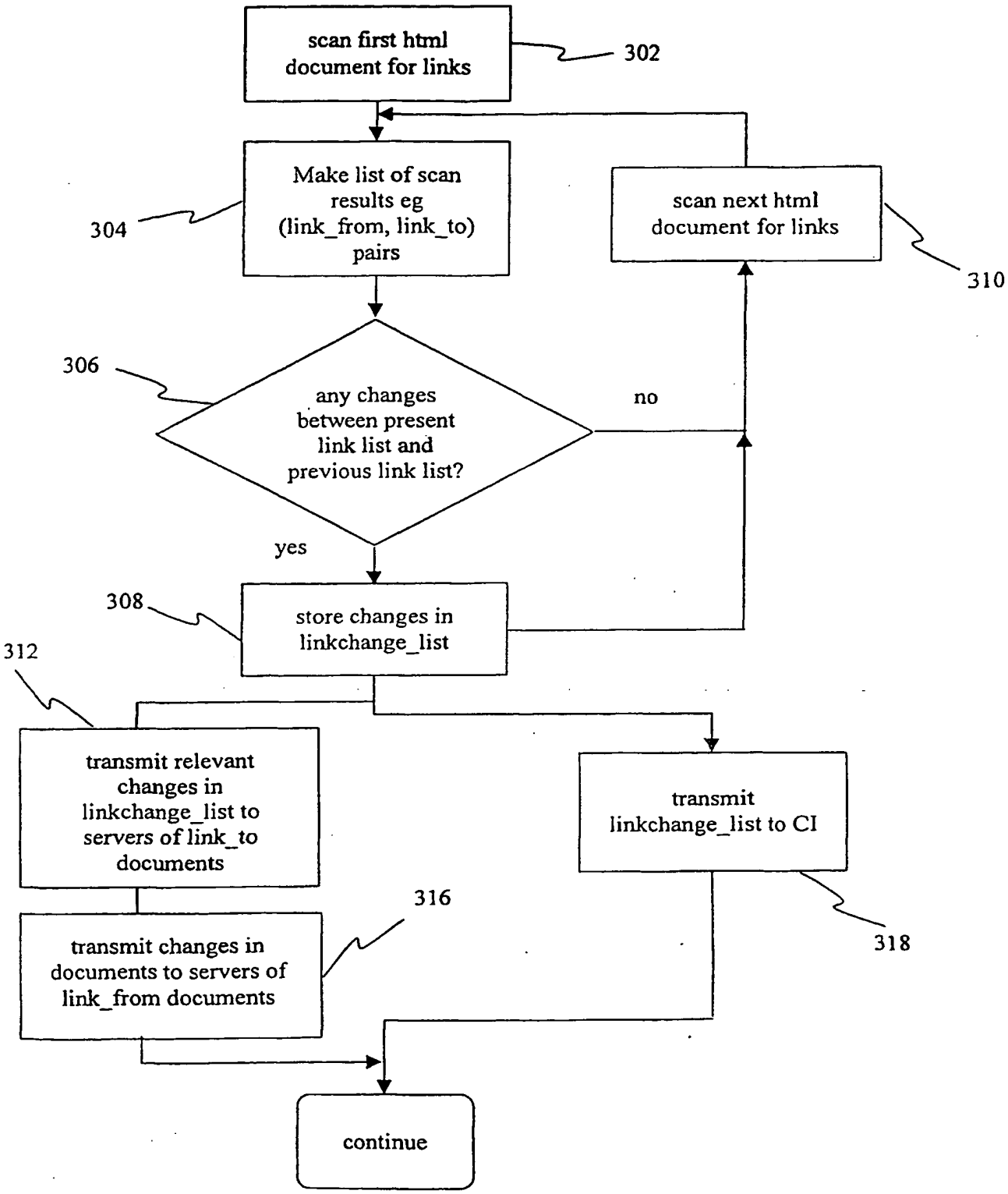
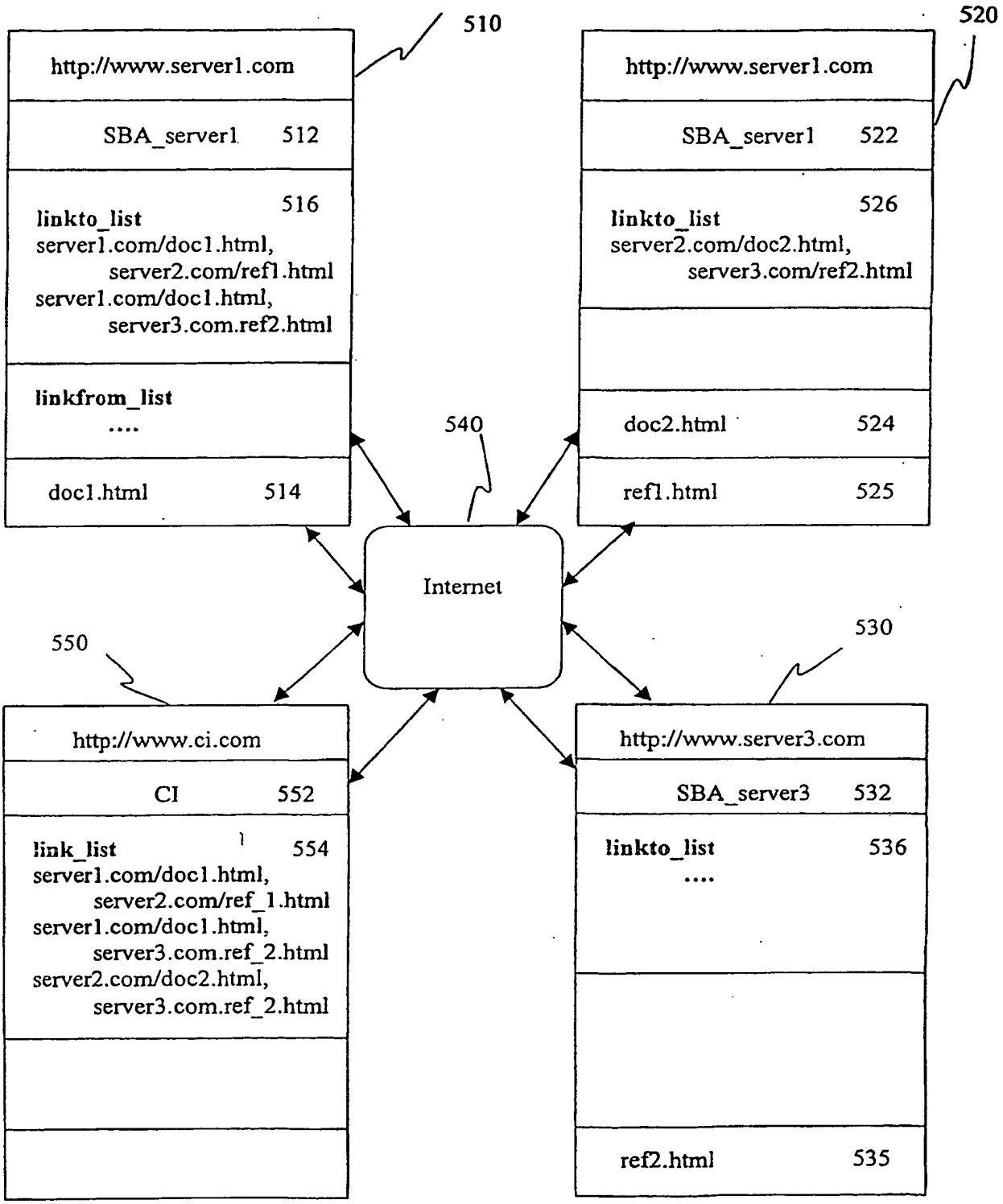


FIG. 1





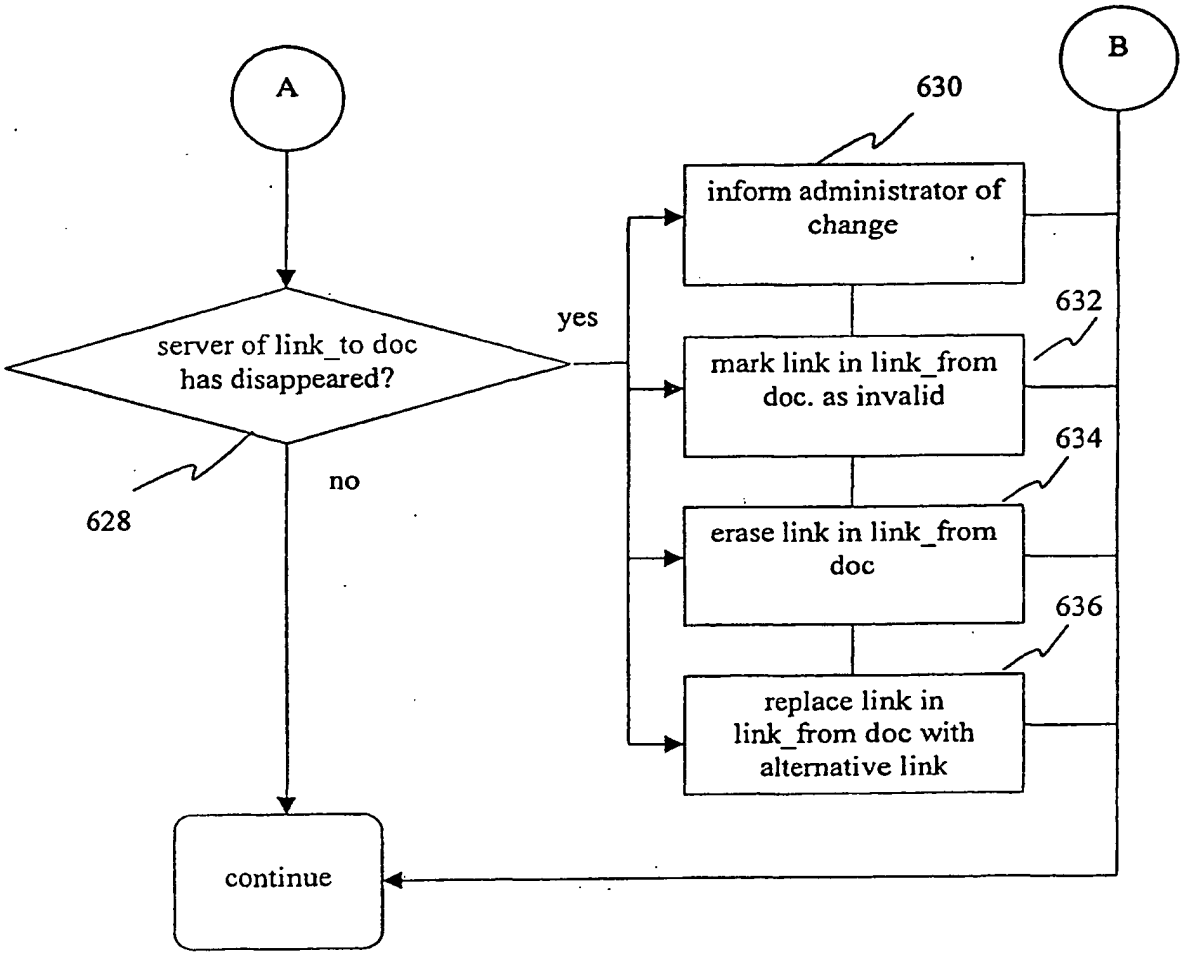


FIG. 6B

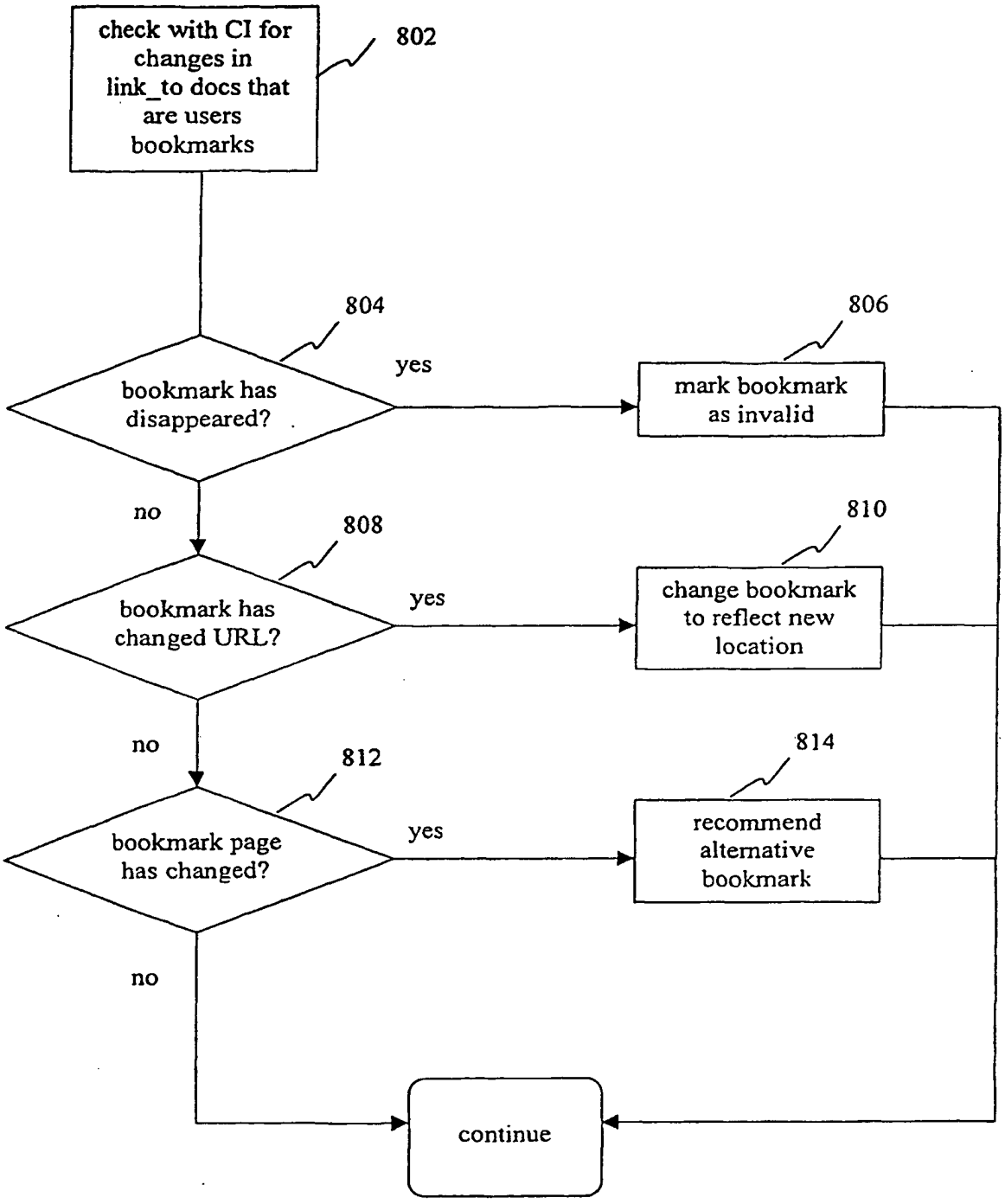


FIG. 8

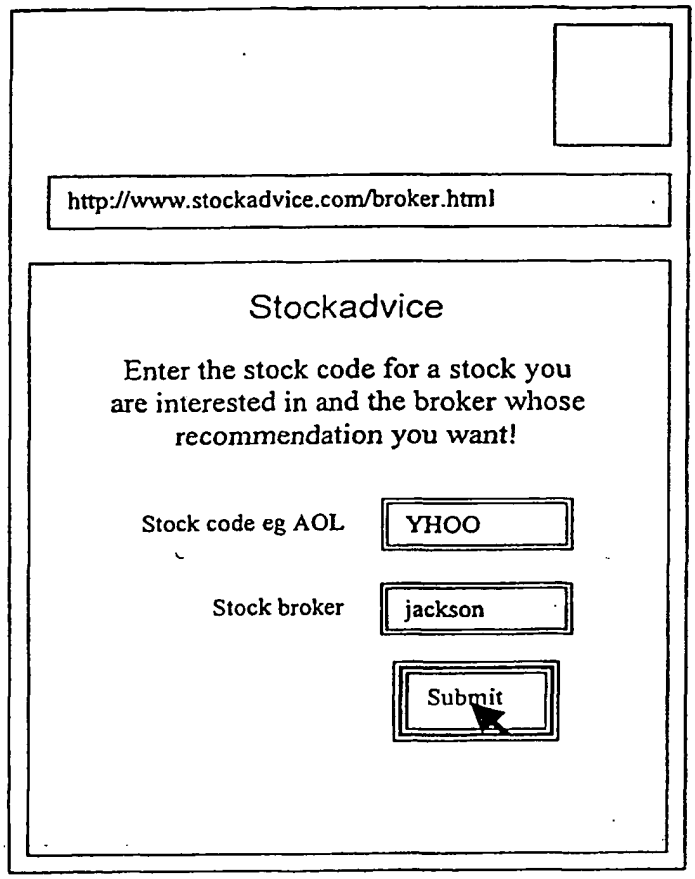


FIG. 9B

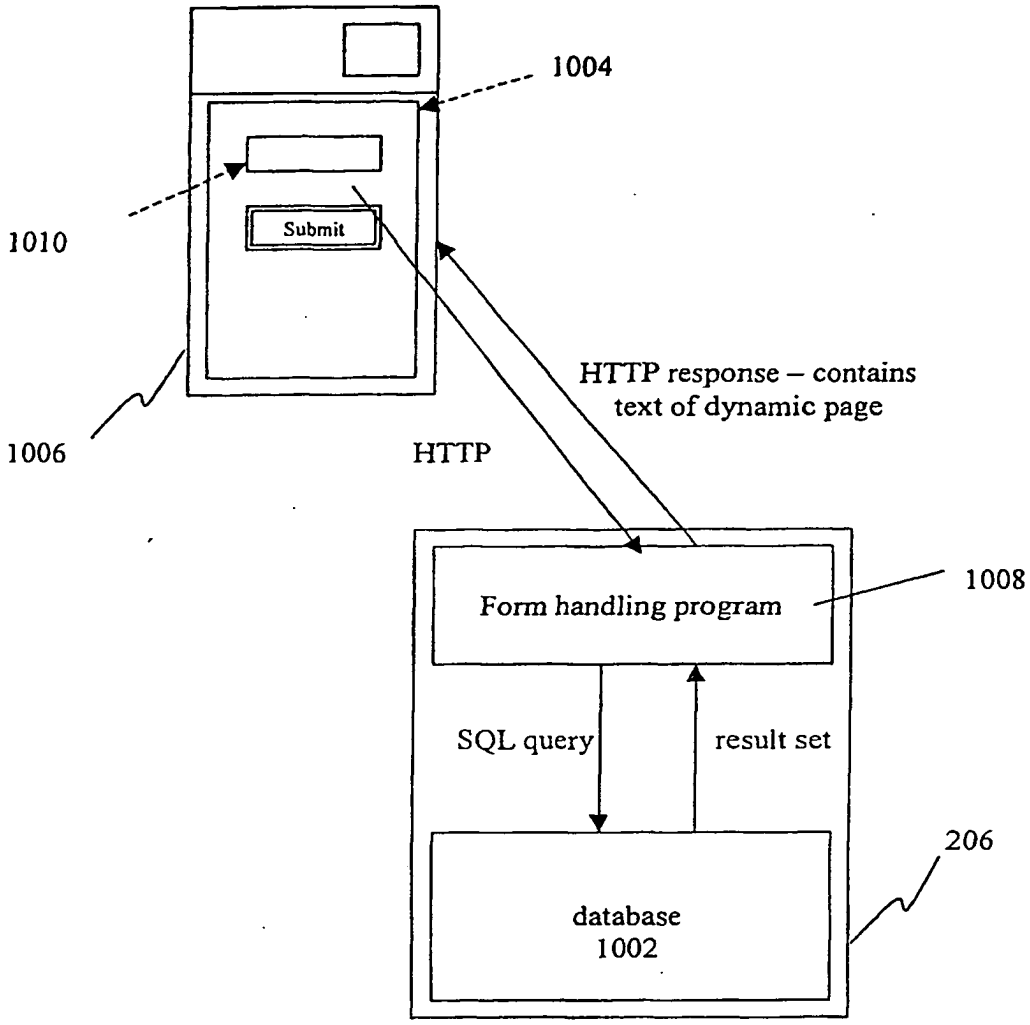


FIG. 10

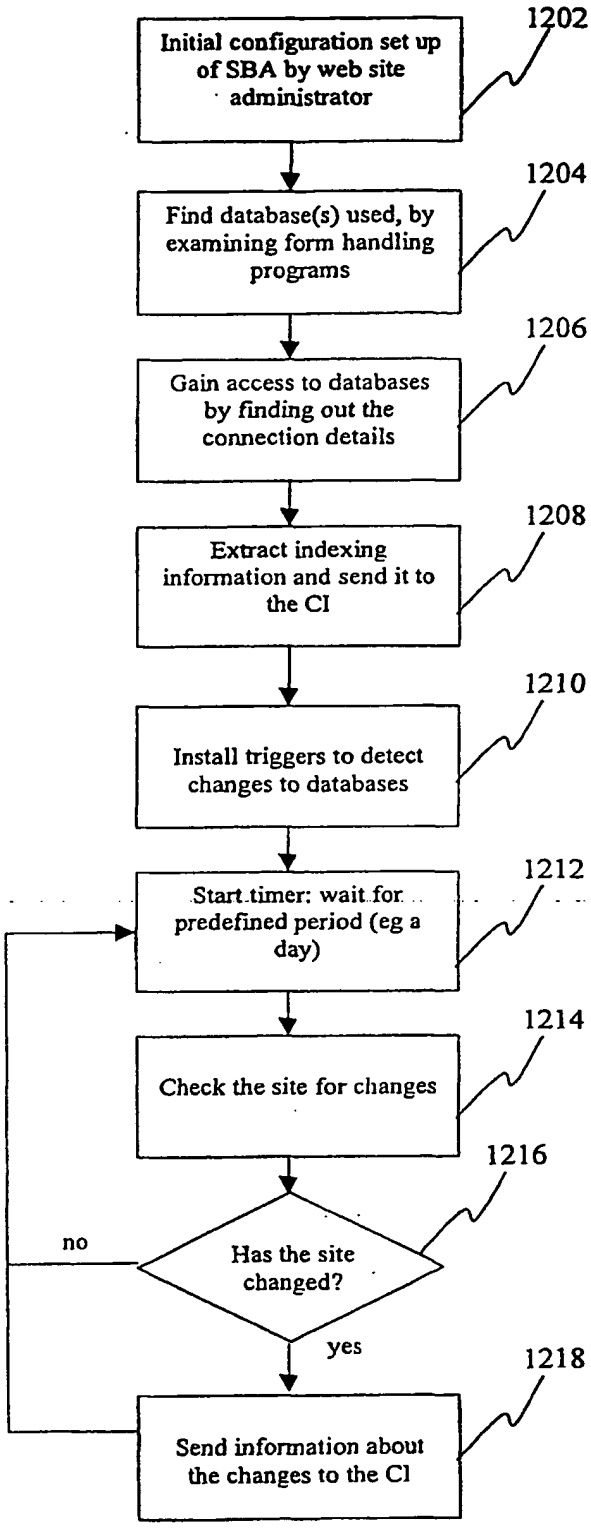


FIG 12.

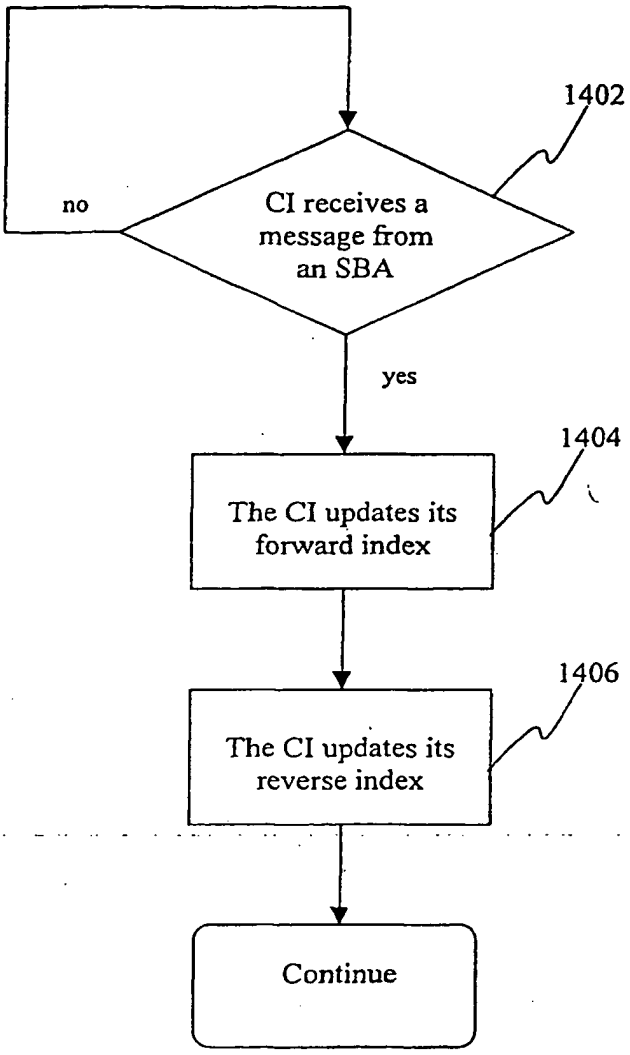


FIG. 14 CI

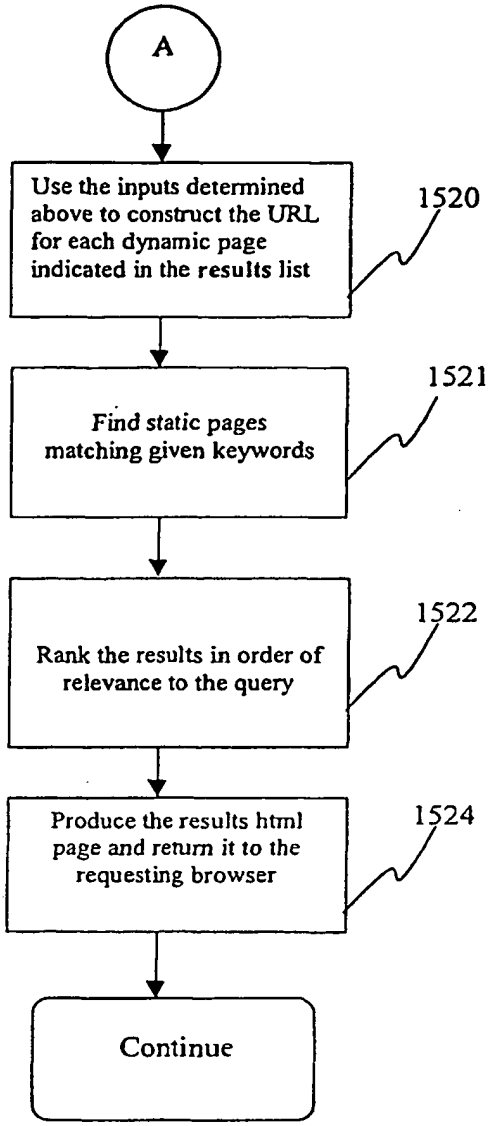


FIG. 15B

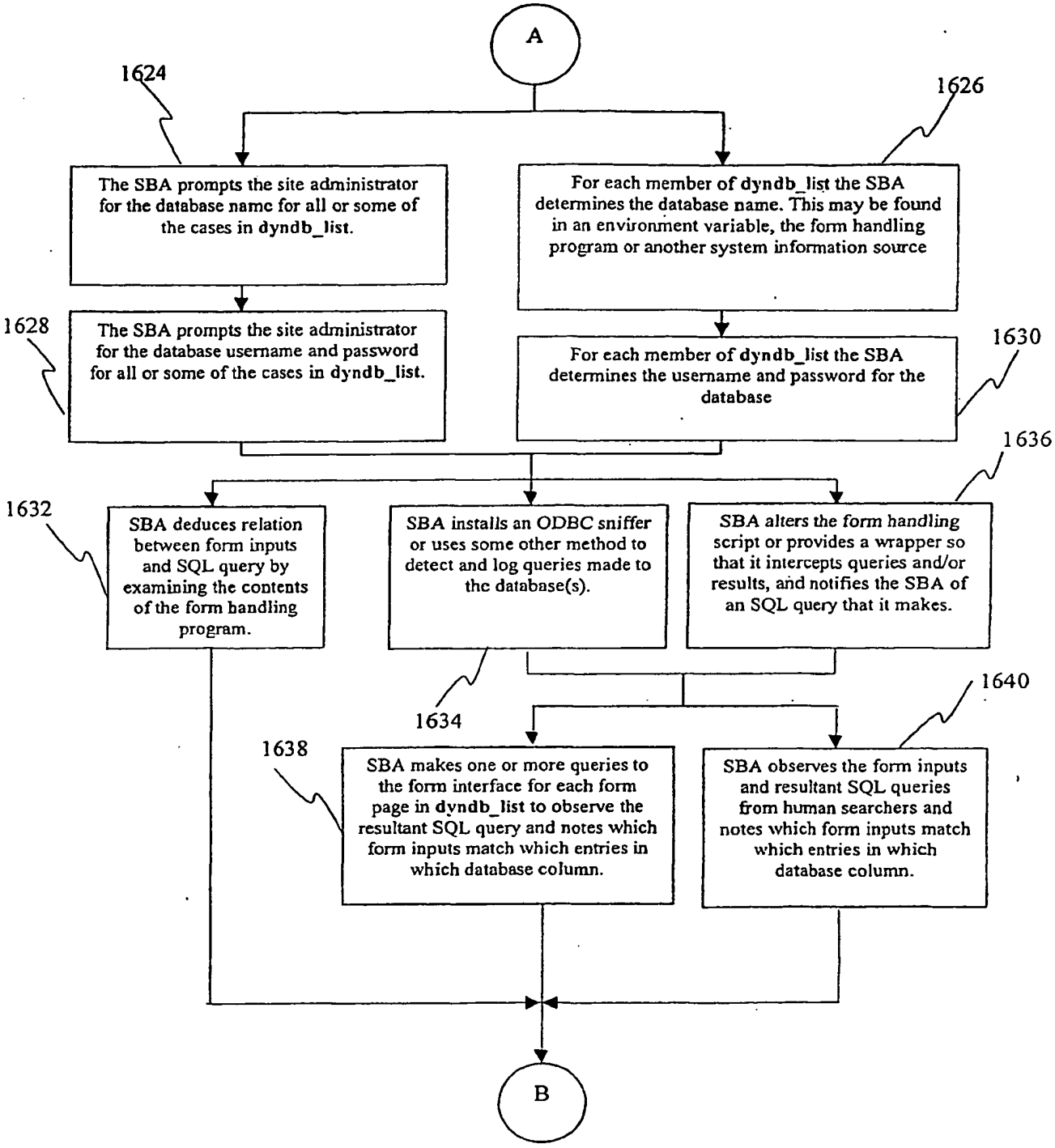


FIG. 16B

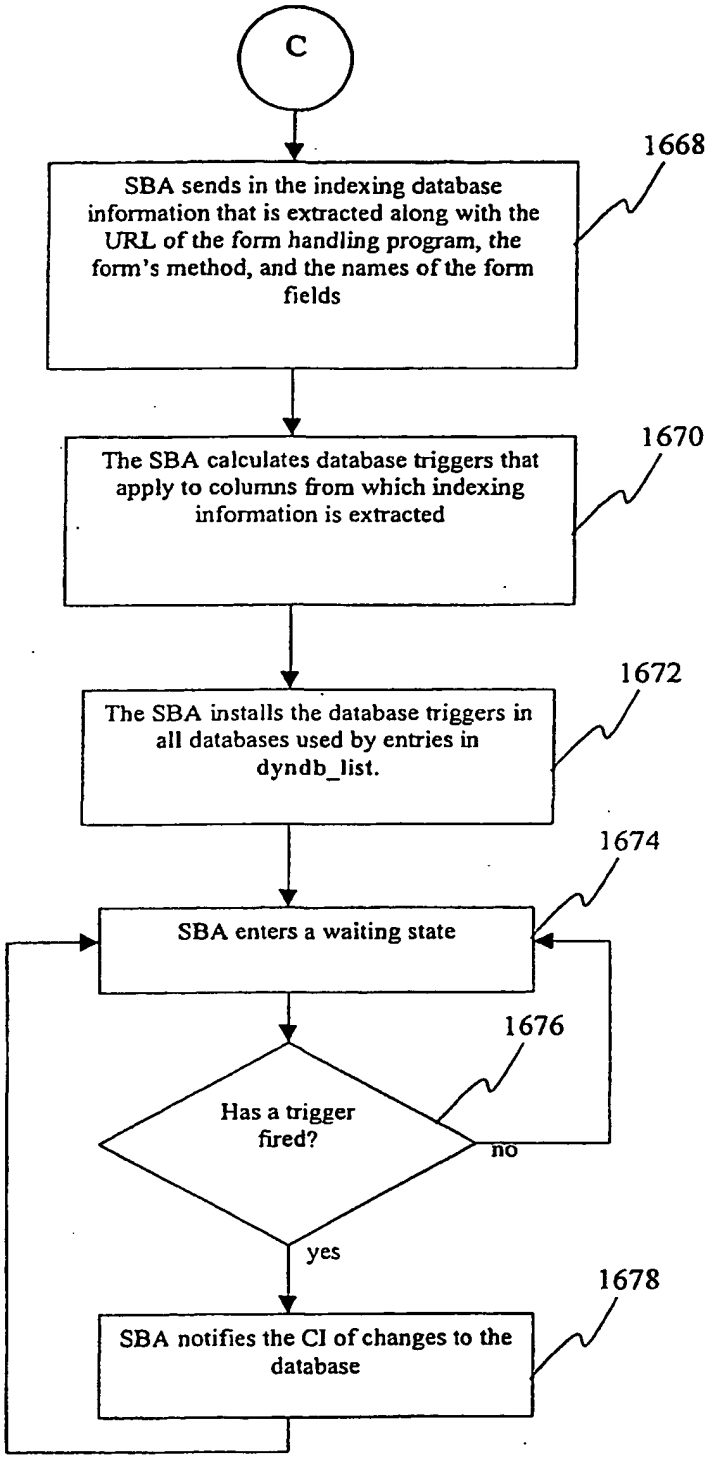


FIG. 16D

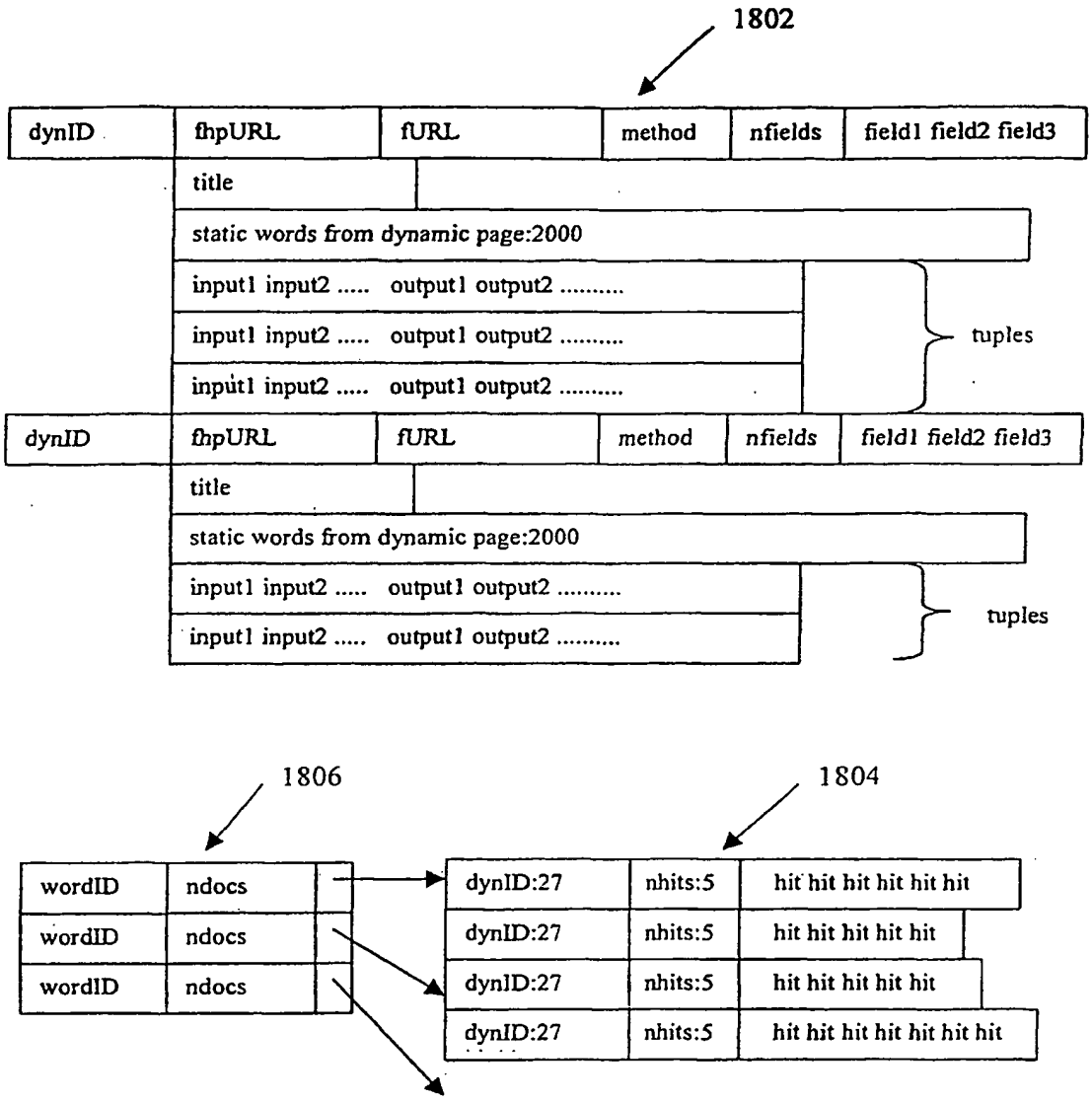


FIG. 18

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU00/01554

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5 649 186 (Ferguson) 15 July 1997 Abstract, column 1, lines 54-58	1
A	US 5 978 799 (Hirsh) 2 November 1999 Abstract, column 1, lines 45-54, col. 2, lines 53-60, column 3, lines 5-10, column 4, lines 13-17	1
A	US 5 864 863 (Burrows) 26 January 1999 Abstract	1
A	US 5 806 063 (Lomet) 8 September 1998 Whole document	1
A	CA 2 209 265 (The Computer Group, Inc.) 27 August 1998 Abstract, figures	1